Lab3 - Mining Data Streams
ID2222 Data Mining

Yage Hao, Hongyi Luo

23 November 2020

1 Introduction

The purpose of this assignment is to implement the algorithm in one of the three selected papers.The paper we chose is the second paper: L. De Stefani, A. Epasto, M. Riondato, and E. Upfal, TRIÈST: Counting Local and Global Triangles in Fully-Dynamic Streams with Fixed Memory Size, KDD'16.
Specifically, we need to implement the graph data stream algorithm based on the reservoir sampling algorithm.And implement the algorithm specifically mentioned in the paper, in our case, it's the improved Triest algorithm.

2 How to Run the Code

To run the code follow the steps above:
Requirements: Jupyter notebook, Python 3.6
1. open the terminal and run
```
> jupyter notebook
```
2. open the file "AssociationRules.ipynb" with the notebook
3. run all the cells

3 Algorithm Explanation

The general idea of the reservoir sampling is if we have a sample pool with length N, and you have an inputting data stream with length M. For every input i, we generate a random number D whose value is between [1, i], if the value is between [1, N]. we use it to replace (or add) into the sample pool [D].

And for the basic_TRIEST, the algorithm can be described as:

Algorithm 1 TRIÈST-BASE

```
    Input: Insertion-only edge stream $\Sigma$, integer $M \geq 6$
 1: $\mathcal{S} \leftarrow \emptyset$, $t \leftarrow 0$, $\tau \leftarrow 0$
 2: for each element $(+, (u, v))$ from $\Sigma$ do
 3:     $t \leftarrow t + 1$
 4:     if SampleEdge($(u, v), t$) then
 5:         $\mathcal{S} \leftarrow \mathcal{S} \cup \{(u, v)\}$
 6:         UpdateCounters($+, (u, v)$)

 7: function SampleEdge($(u, v), t$)
 8:     if $t \leq M$ then
 9:         return True
10:     else if FlipBiasedCoin($\frac{M}{t}$) = heads then
11:         $(u', v') \leftarrow$ random edge from $\mathcal{S}$
12:         $\mathcal{S} \leftarrow \mathcal{S} \setminus \{(u', v')\}$
13:         UpdateCounters($-, (u', v')$)
14:         return True
15:     return False

16: function UpdateCounters($(\bullet, (u, v))$)
17:     $\mathcal{N}^{\mathcal{S}}_{u,v} \leftarrow \mathcal{N}^{\mathcal{S}}_{u} \cap \mathcal{N}^{\mathcal{S}}_{v}$
18:     for all $c \in \mathcal{N}^{\mathcal{S}}_{u,v}$ do
19:         $\tau \leftarrow \tau \bullet 1$
20:         $\tau_c \leftarrow \tau_c \bullet 1$
21:         $\tau_u \leftarrow \tau_u \bullet 1$
22:         $\tau_v \leftarrow \tau_v \bullet 1$
```

For the improved_TRIEST, the algorithm can be described as:

Algorithm TRIÈST-IMPR

```
    Input: Insertion-only edge stream Σ, integer M ≥ 6
 1: S ← ∅, t ← 0, τ ← 0
 2: for each element (+, (u, v)) from Σ do
 3:     t ← t + 1
 4:     UpdateCounters(+, (u, v), t)
 5:     if SampleEdge((u, v), t) then
 6:         S ← S ∪ {(u, v)}

 7: function SampleEdge((u, v), t)
 8:     if t ≤ M then
 9:         return True
10:     else if FlipBiasedCoin(M/t) = heads then
11:         (u0, v0) ← random edge from S
12:         S ← S \ {(u0, v0)}
13:         return True
14:     return False

15: function UpdateCounters(•, (u, v), t)
16:     N_{u,v}^S ← N_u^S ∩ N_v^S
17:     delta = max{1, (t - 1)(t - 2)/M(M - 1)}
18:     for all c ∈ N_{u,v}^S do
19:         τ ← τ • delta
20:         τ_c ← τ_c • delta
21:         τ_u ← τ_u • delta
22:         τ_v ← τ_v • delta
```

4 Code Explanation

We construct the code into three part.

Part 0 is the preprocessing. In which we import the whole dataset and in order to ease our test process, we form a dataset by selecting the first 10000 elements from the original one. (dataset: https://snap.stanford.edu/data/web-Stanford.html)

Part 1 is our implementation of TRIEST-BASE algorithm.
We implement a large function TRIEST_BASE(M, dataset=dataset). This function estimates number of global triangles by TRIEST_BASE algorithm. It contains the following argument: M: size of reservoir sample, M>=6 and dataset, =dataset defaultly. It returns number of global triangle count.
In the TRIEST_BASE function, there are two sub-functions. First one is sample_edge(edge, t, S, tau, M=M). This function is the reservoir sampling process. Notice that each edge item in the sample has equal probability. It returns True or False: whether the input edge will replace an existing edge in the edge sample.
Another one is update_counters(edge, S, tau, t, M=M). This function compute neighborhood of vertices and update global and local counters. It returns tau: updated global counter.

Part 2 is our implementation of TRIEST-IMPR algorithm which has similar constructure as TRIEST-BASE part.

5 Results

We tested different M value, and got the results showed below:

TRIEST-BASE:

True Vlue	M	Global Estimations
30190.0	1000	61165.07411218834
	3000	37655.9891144502
	5000	36290.88835534214
	7000	33878.99204053664
	9000	30470.151635973678

TRIEST-IMPR

We first set M=5000 in the above improved sampling case and the size of the dataset we use is 10000.
We can see that when t<=M, our estimated number of triangles by TRIEST_IMPR algorithm is exactly the same as true value.

```
Compute true value:
element index = 0, tau = 0
element index = 1000, tau = 37
element index = 2000, tau = 296
element index = 3000, tau = 1002
element index = 4000, tau = 2367
element index = 5000, tau = 4555
element index = 6000, tau = 7557
element index = 7000, tau = 11411
element index = 8000, tau = 16712
element index = 9000, tau = 22707
True value is 30190.

Compute estimate value by reservoir sampling:
element index = 0, tau = 0
element index = 1000, tau = 37
element index = 2000, tau = 296
element index = 3000, tau = 1002
element index = 4000, tau = 2367
element index = 5000, tau = 4555
element index = 6000, tau = 7610.927455811263
element index = 7000, tau = 11942.63614002814
element index = 8000, tau = 18017.876664293035
element index = 9000, tau = 25427.016077535725
The Estimated value is 35295.12952070433.

Error: 5105.1295207043295 triangles.
Error rate: 0.16910001724757634.
```

We run 20 times the TRIEST_IMPR algorithm with M=5000. Comparing results with the ones obtained by the TRIEST_BASE algorithm, we can see that the variance of the improved algorithm is lower and the mean of the improved algorithm is closer to the true value as we expected.

True value	3019.0
mean from TRIEST_BASE	38042.21380552221
mean from TRIEST_IMPR	35072.02691547922
variance from TRIEST_BASE	3170998.664030402

variance from TRIEST_IMPR	423135.8354098646

5 Extra questions

5.1 What were the challenges you have faced when implementing the algorithm?

The main challenges we faced is how to decide the optimal value of M. Cause this value mainly influences if the model is stable. Of course, understanding the literature provided is also quite a challenge.

5.2 Can the algorithm be easily parallelized? If yes, how? If not, why? Explain.

It can be. The local estimation can be executed in different computing nodes then they can be gathered and generate the final results.

5.3 Does the algorithm work for unbounded graph streams? Explain.

It certainly can be applied in that. Because even if the graph stream is unbounded, it is for an infinite time. For every certain moment, the inputting data stream length is still fixed. So we can also estimate the target numbers which hidden in the stream. However, for unbounded streams, it is impossible to count the total number of triangles until all the edges have been received. Some technique like sliding windows should be used to output rolling estimates for given time periods. And for unbounded streams, M should be carefully chosen since it will determine the estimation accuracy and is related to data size.

5.4 Does the algorithm support edge deletions? If not, what modification would it need? Explain.

We used TRIÈST-IMPR instead of TRIÈST-FD which does not support edge deletions for streaming graph. The reason why we choose TRIÈST-IMPR lies in that the data streaming for this lab doesn't require any delete operation so we haven't implement the edge deletion function. To modify the algorithm for future edge delete function, random pairing technique should be used. We need to add two more counters, di and do, to keep track of the number of uncompensated edge deletions, which involves an edge e that was in S at the time the deletion for e was on the stream